МИНОБРНАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное
учреждение высшего профессионального образования
«Санкт-Петербургский политехнический
университет Петра Великого»
Институт компьютерных наук и технологий
Высшая школа искусственного интеллекта

Отчет о курсовой работе

Дисциплина: «Промышленное программирование на Python»

Сдал: _____ Черников С. Г. группа 3540201/20201
Принял: _____ к.т.н. ассист. Моторин Д.Е.

Санкт-Петербург, 2023

Содержание

Введение

Мобильные роботы, широко применяющиеся в различных областях, являются перспективной областью для разработок. Одной из технологий в данной тематике является задача планирования пути, связанная с поиском маршрута между начальным и конечным состоянием с учетом заданного критерия оптимизации и ограничений.

На сегодняшний день эта задача активно изучается, и для ее решения предложено множество подходов. Так, алгоритм потенциальных полей (artificial potential field algorithm, APF) основан на моделировании движущегося агента как частицы в потенциальном поле; конечная точка обладает минимальным потенциалом, которого и стремится достичь частица, в то же время вокруг каждого препятствия потенциал повышается и убывает с удалением от препятствий, в результате чего частица стремится их облетать, двигаясь в сторону цели. Метод дает плавные траектории, однако может попасть в точку локального минимума потенциала.

Бионические алгоритмы, вдохновленные самоорганизующимися природными системами, такие как генетические или муравьиные алгоритмы, хорошо применимы, но ограничены низкой скоростью работы и могут привести к локально оптимальному решению. Алгоритмы поиска по решетке (обычно, A*) требуют дискретизируемого пространство небольшой размерности.

Алгоритмы, основанные на сэмплировании, такие как RRT, хорошо применимы для пространств большой размерности и не требуют явного моделирования пространства. Однако RRT лишь вероятностно полон, имеет низкую скорость сходимости и может дать неоптимальный результат. Существуют различные модификации RRT, которые будут рассмотрены далее. Предлагаемое решение основано на нескольких модификациях RRT: быстром RRT-Connect и RRT*, который значительно оптимизирует путь. В решении используется стратегия эвристического сэмплирования из определенных областей, что повышает скорость сходимости и использование взятых узлов.

1 Постановка задачи

Пусть $X \subset R^d$ – конфигурационное пространство размерности d, $d \geq 2$; $X_{obs} \subset X$ – множество препятствий, $X_{free} = X$
X_{obs} – пустое пространство. Начальная и конечная точки $x_{start}, x_{goal} \in X_{free}$. Путь отражает непрерывная функция $\sigma : [0,\ 1] \to X$. Если $\forall \tau \in [0,\ 1]\ \ \sigma(\tau) \in X_{free}$, путь σ не пересекает препятствия, и задача состоит в нахождении такого пути между точками x_{start}, x_{goal}, называемого достижимым. Множество достижимых путей $\Sigma = \{\, \sigma : [0,\ 1] \to X_{free} \mid \sigma(0) = x_{start},\ \ \sigma(1) = x_{goal} \,\}$. Вводится функция оценки стоимости пути $c(\sigma)$.

Задача планирования оптимального пути состоит в нахождении достижимого пути σ^*, имеющего минимальную стоимость:

$$\sigma^* = \arg \min_{\sigma \in \Sigma} c(\sigma)$$

2 Результаты теоретических исследований

2.1 Метод решения

Недостатком RRT* является тот факт, что он требует большое число операций, чтобы найти оптимальный путь. Проблема отчасти решается в Informed RRT*, однако используемый им многомерный эллипсоид не дает желаемого эффекта в случае, если имеется узкий коридор, через который должен пройти оптимальный маршрут.

Для дальнейшего увеличения скорости сходимости алгоритма в работе [1] предложен метод EP-RRT*. Перед тем, как найти начальное (неоптимальное) решение, алгоритм работает схожим с RRT-Connect образом. Как только найдено начальное решение (неоптимальный путь), сэмплирование происходит эвристически в определенной области. Ниже приведен псевдокод алгоритма EP-RRT*.

$T_{init} \leftarrow RRT - Connect;\ T \leftarrow T_{init};$
$\sigma_{init} \leftarrow GetPath\,(T)\,;$
$X_{expand} \leftarrow Expand\,(\sigma_{init})\,;$
$for\ i = 1\ to\ N\ do$
    $x_{rand} \leftarrow HeuristicSample\,(X_{expand})\,;$
    $x_{nearest} \leftarrow Nearest\,(V, x_{rand})\,;$
    $x_{new} \leftarrow Steer\,(x_{rand}, x_{nearest})\,;$
    $if\ CollisionFree\,(x_{new}, x_{nearest})\ then$
        $X_{near} \leftarrow Near\,(T, x_{new}, \eta)\,;$
        $x_{parent} \leftarrow ChooseParent\,(X_{near}, x_{nearest}, x_{new})\,;$
        $V \leftarrow V \cup \{x_{new}\}; E \leftarrow E \cup \{(x_{parent}, x_{new})\};$
        $T \leftarrow Rewire\,(T, x_{new}, X_{near})\,;$
        $\sigma \leftarrow GetPath\,(T)\,;$
        $X_{expand} \leftarrow Expand\,(\sigma)\,;$
    $end\ if$
$end\ for$
$return\ T;$

Алгоритм имеет два основных улучшения по сравнению с RRT*. Первое состоит в использовании жадной стратегии RRT-Connect для быстрого нахождения пути без коллизий. Второе вдохновлено идеей области многомерного эллипсоида из алгоритма Informed RRT* и также состоит в построении области сэмплирования для повышения скорости сходимости. В сложных средах, таких как узкие коридоры или лабиринты, лучшие варианты сэмплирования распределены вокруг углов или краев, что учитывается алгоритмом при выборе области сэмплирования.

Главным аспектом эвристического сэмплирования является определение области для него. Поскольку путь представляет собой набор вершин, последовательно соединенных отрезками, область для расширения представляется как набор соединенных четырехугольников. На рисунке 1 приведен пример пути из четырех вершин и трех соединяющих их ребер, а область расширения составлена из восьми вершин, полученных из имеющихся.

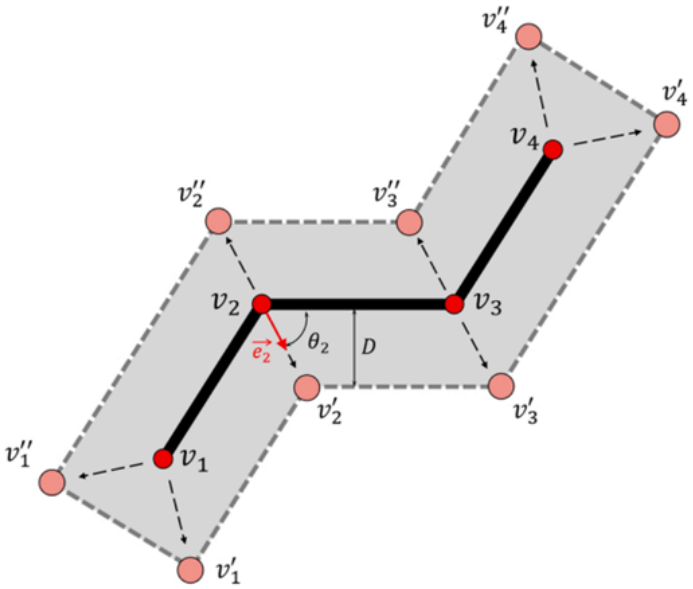

Рис. 1: Расширение пути

Для вершины v_i вектор биссектрисы $\vec{e_i}$ между двумя смежными зонами можно получить как:

$$\vec{e_i} = \frac{v_i\vec{v_{i-1}}}{|v_i\vec{v_{i-1}}|} + \frac{v_i\vec{v_{i+1}}}{|v_i\vec{v_{i+1}}|}$$

Две новые вершины для области расширения пути получаются на основе параметра дистанции расширения D как:

$$v'_i = v_i + \frac{\vec{e_i}}{|\vec{e_i}|}\frac{D}{cos\theta_i}$$

$$v''_i = v_i - \frac{\vec{e_i}}{|\vec{e_i}|}\frac{D}{cos\theta_i}$$

где θ_i – половина угла поворота пути вокруг вершины v_i.

Далее приведен псевдокод алгоритма процесса расширения.

$\{v_1, v_2, \ldots, v_{n-1}, v_n\} \leftarrow \sigma; V_{ex} \leftarrow \emptyset;$
$for\ i = 2\ to\ n\ \breve{}1\ do$
$\quad \vec{e}_i \leftarrow DirectionVector\,(v_{i-1}, v_i, v_{i+1})\,;$
$\quad \{v_i', v_i''\} \leftarrow Extension\,(\vec{e}_i, v_i)\,;$
$\quad V_{ex} \leftarrow V_{ex} \cup \{v_i', v_i''\};$
$end\ for$
$\{v_1', v_1'', v_n', v_n''\} \leftarrow ExtensionEnd\,(v_1, v_i)\,;$
$V_{ex} \leftarrow V_{ex} \cup \{v_1', v_1'', v_n', v_n''\};$
$X_{expand} \leftarrow Encircle\,(V_{ex})\,;$
$return\ X_{expand};$

В процессе расширения для оптимизации ранее найденного пути важным аспектом является выбор параметра дистанции расширения D, который предлагается изменять со временем. Начальная величина параметра D_{base} основана на размере исследуемой области пространства конфигураций.

$$D_{base} = \frac{max\{L, W\}}{\epsilon}$$

где ϵ – начальный коэффициент расширения (настраиваемый, в работе используется 8), L, W – соответственно, длина и ширина области. Вводится динамически изменяемый коэффициент расширения k:

$$k = \frac{1}{2\pi} arccot\left((i - i_{init}) - \frac{N - i_{init}}{2}\right) + 0.75$$

Здесь i – номер текущей итерации, i_{init} – номер итерации, на которой алгоритм нашел начальный путь с помощью RRT-Connect, N – максимальное число итераций. Таким образом, итоговый параметр расширения D вычисляется как:

$$D = kD_{base}$$

2.2 Результаты экспериментов

В работе было смоделировано 4 различные среды, в которых тестировались алгоритмы RRT*, Informed RRT*, EP-RRT*.

Результаты показали, что EP-RRT* показывает лучший результат в каждой из сред – он обладает лучшей скоростью сходимости, и сходится к более оптимальному решению, в особенности, в тех средах, которые устроены достаточно сложно и в которых присутствуют узкие коридоры или лабиринты.

Примечательно, что метод Informed-RRT*, который так же предлагает ограничить область сэмплирования, работает в таких средах крайне плохо. Это можно объяснить тем, что EP-RRT* позволяет сэмплировать точки в наиболее важных для оптимизации областях, расположенных именно вокруг текущего оптимального пути.

Метод обладает высокой стабильностью, и при большом числе запусков алгоритма дает малую дисперсию в скорости получения оптимального или субоптимального решения.

3 Описание использованного программного обеспечения

Для программной реализации алгоритма был использован язык программирования Python 3.9. В качестве основы была использована реализация семейства алгоритмов RRT из репозитория https://github.com/motion-planning/rrt-algorithms. Эта реализация, в свою очередь, использует низкоуровневую библиотеку Rtree для работы с геометрическими объектами, которая реализована на языке программирования C и представляет собой оболочку, вызываемую из кода на Python.

Вследствие этого, выбор языка целесообразен, поскольку не ведет к значительному снижению быстродействия решения и позволяет использовать преимущества высокоуровневого языка Python.

Были также использованы библиотеки numpy для реализации алгебраических операций и plotly/matplotlib для визуализации результатов.

4 Реализация программы

Программная реализация заключалась в доработке кода, представленного в репозитории https://github.com/motion-planning/rrt-algorithms.

Был унаследован класс, реализующий RRT* алгоритм, поскольку метод EP-RRT* использует и логику работы RRT* для оптимизации путей после сэмплирования.

В классе-наследнике был реализован метод непосредственно выполняющий поиск пути с помощью EP-RRT. Код метода приведен далее.

Входные данные: пространство поиска, множество препятствий, начальная и конечная точка, максимальное число сэмплирований до остановки алгоритма, минимальная толщина препятствия, число соседних узлов для оптимизации RRT*, вероятность проверки достижимости конечной точки после данного сэмплирования.

Выходные данные: построенный путь в виде последовательности вершин.

```
def ep_rrt_star(self, optimize: bool =True, epsilon: int =8):
    default_path = self.rrt_connect.rrt_connect()

    for i in range(len(default_path)):
        self.add_vertex(0, default_path[i])
        if i != 0:
            self.add_edge(0, default_path[i], default_path[i - 1])

    best_path = default_path
    while True:
        for q in self.Q:  # iterate over different edge lengths
            for i in range(q[1]):  # iterate over number of edges of given length to add
                D = np.max(self.X.dimension_lengths) / epsilon
                x_new, x_nearest = self.new_and_near(0, q, zone=Zone(best_path, D))
                if x_new is None:
                    continue

                # get nearby vertices and cost-to-come
                L_near = self.get_nearby_vertices(0, self.x_init, x_new)
```

```
20
21                  # check nearby vertices for total cost and
    connect shortest valid edge
22                  self.connect_shortest_valid(0, x_new,
    L_near)
23
24                  if x_new in self.trees[0].E:
25                      # rewire tree
26                      self.rewire(0, x_new, L_near)
27
28                  solution = self.check_solution()
29                  if solution[0] and (not optimize or self.
    samples_taken >= self.max_samples):
30                      return solution[1]
31
32                  best_path = self.reconstruct_path(0, self.
    x_init, self.x_goal)
33                  logging.info(f"{self.samples_taken} {self.
    __calc_path_len(best_path)}")
```

В методе предлагается построить неоптимальный путь при помощи алгоритма RRT-Connect, после чего следует цикл из сэмплирований для его дальнейшей оптимизации.

Также введен класс Zone, инициализирующийся на основе передаваемого ему пути, вокруг которого и должна построиться зона сэмплирования.

Для сэмплирования предлагается не строить в явном виде многоугольник, ограничивающий зону, а реализовать алгоритм, который выбирает случайное число – длину базовой точки в оптимизируемом пути, случайное число – отклонение от пути в определенную сторону, не больше параметра ширины зоны D, и определенный угол, на который точка может повернуться. Реализация метода сэмплирования приведена далее.
Входные данные: оптимизируемый путь, вокруг которого строится зона сэмплирования; ширина зоны сэмплирования.
Выходные данные: случайная точка в зоне сэмплирования, подчиняющаяся равномерному распределению.

```
1 def random(self) -> np.ndarray:
2     l = np.random.uniform(size=1)[0] * self.path_len
3
4     slen = 0
5     for i in range(len(self.lens)):
6         if slen + self.lens[i] >= l:
```

```
7                break
8            slen += self.lens[i]
9
10       delta = l - slen
11       base = self.path[i] + (self.path[i + 1] - self.path[i])
         * delta / self.lens[i]
12
13       if i > 0:
14           theta = self.__angle(self.path[i - 1] - self.path[i
         ], self.path[i + 1] - self.path[i]) / 2
15       else:
16           theta = 0
17
18       angle = np.random.uniform(-1, 1, size=1)[0] * theta
19       random_space = np.random.uniform(-self.D, self.D, size
         =1)[0]
20
21       base_unrotated = base + random_space * (self.path[i +
         1] - self.path[i]) * max(np.tan(angle), 1)
22       x, y = base_unrotated - base
23       if x == 0:
24           x = 1e-2
25       vector_rotation = np.array([-y / x, 1.])
26       vector_rotation /= np.linalg.norm(vector_rotation) *
         random_space
27       rotated = base + vector_rotation
28
29       return rotated
```

5 Анализ работоспособности решения

Для запуска алгоритма использовалась двумерная среда. Среда генерировалась случайно при помощи алгоритма, приведенного также в репозитории https://github.com/motion-planning/rrt-algorithms.

Эксперимент предполагал запуск алгоритмов RRT* и EP-RRT*, сравнение деревьев, построенных алгоритмами, и графиков скорости их сходимости.

На рисунках 2, 3 приведены деревья, построенные алгоритмами RRT* и EP-RRT* соответственно.

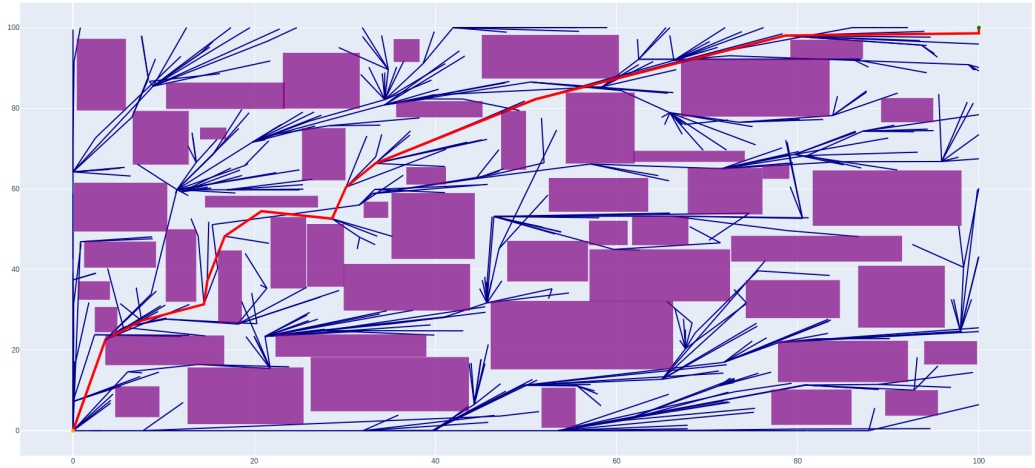

Рис. 2: Результаты работы алгоритма RRT*

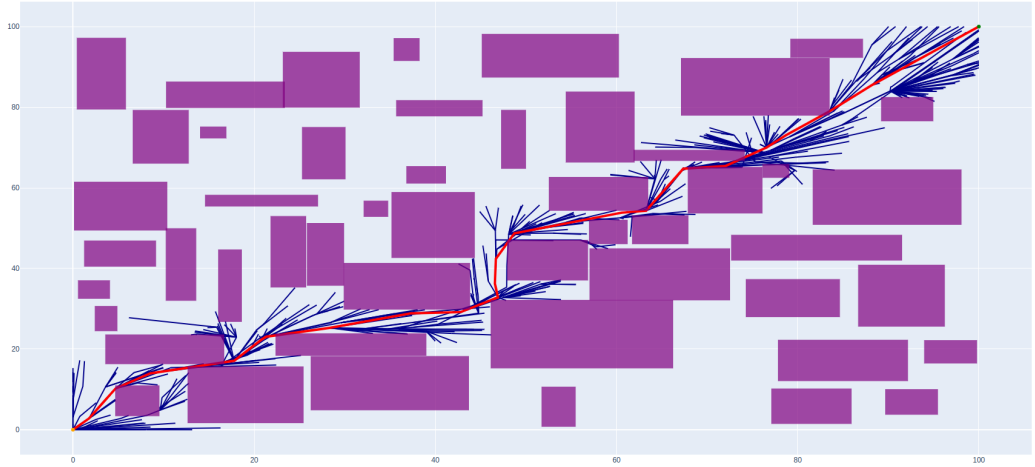

Рис. 3: Результаты работы алгоритма EP-RRT*

Можно видеть, что RRT* хаотично сэмплирует точки в каждую сторону по всему пространству, в то время как EP-RRT* сэмплирует лишь в узкой области, близкой к оптимизируемому пути.

На рисунке 4 приведены графики зависимости номера итерации от длины найденного оптимального пути для алгоритмов RRT* и EP-RRT*.

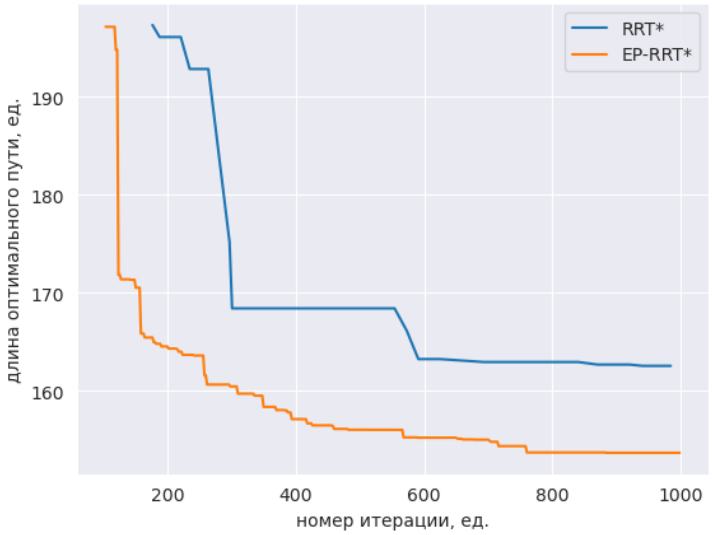

Рис. 4: Сравнение алгоритмов RRT* и EP-RRT*

Можно видеть, что EP-RRT* раньше нашел первое решение, поскольку использовал RRT-Connect, и далее сходился быстрее и к более оптимальному решению, нежели RRT*.

Выводы

Алгоритм EP-RRT* предложен для того, чтобы устранить недостатки RRT*, такие как низкая скорость сходимости и лишние ветви в особых средах. Основной идеей алгоритма является использование жадного поиска неоптимального пути с помощью RRT-Connect и дальнейшая оптимизация найденного решения путем рассмотрения определенной окрестности найденного маршрута.

Предложенный метод значительно увеличивает вероятность сэмплирования наиболее важных точек, что повышает скорость сходимости. Метод может быть совмещен с любым алгоритмом, основанном на сэмплировании.

Результаты экспериментов показывают, что предложенный метод по сравнению с прочими быстрее сходится и дает более оптимальное решение, в особенности, в случае узких коридоров и лабиринтов. Это было продемонстрировано и в данной работе.

Однако несмотря на преимущества метода, он имеет свои ограничения в производительности. В средах, где есть множество альтернативных путей различной стоимости, алгоритм требует области расширения больших размеров, чтобы избежать попадания в локальный оптимум. Действительно, если RRT-Connect нашел бы крайне неоптимальный путь, очень далекий от оптимального, например, идущий с другой стороны от крупного препятствия, то для выхода из локального оптимума требуется очень широкая область сэмплирования, которая бы охватила это препятствие с другой стороны.

Дальнейшие исследования могут быть направлены на изучение возможности адаптивной настройки параметров предложенного алгоритма.

Список литературы

1. An improved RRT* algorithm for robot path planning based on path expansion heuristic sampling / J. Ding [и др.] // Journal of Computational Science. — 2023. — Т. 67. — С. 101937. — ISSN 1877-7503. — DOI: https://doi.org/10.1016/j.jocs.2022.101937. — URL: https://www.sciencedirect.com/science/article/pii/S1877750322002964.